ICASE

HIGHLY PARALLEL MULTIGRID SOLVERS FOR ELLIPTIC PDEs: AN EXPERIMENTAL ANALYSIS

Dennis Gannon

and

John Van Rosendale

(NASA-CR-185813) HIGHLY PARALLEL MULTIGRID SOLVERS FOR ELLIPTIC PDEs: AN EXPERIMENTAL ANALYSIS (ICASE) 40 p

N89-71362

Unclas

00/64 0224358

Report No. 82-36

November 17, 1982

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING

NASA Langley Research Center, Hampton, Virginia 23665

Operated by the

UNIVERSITIES SPACE USRA RESEARCH ASSOCIATION

Highly Parallel Multigrid Solvers for Elliptic PDEs: An Experimental Analysis.

Dennis Gannon

Department of Computer Sciences, Purdue University

John Van Rosendale

ICASE, NASA Langley Research Center

ABSTRACT

Computer Architectures consisting of many thousands of processing elements have been proposed and studied in the literature. In many cases, the motivation for building these systems is to speed-up large scale scientific computation such as the solution of elliptic partial differential equations. To provided a basis for comparing architectural alternatives, it is helpful to analyze the various classes of algorithms that might be well suited to a highly parallel implementation. This paper considers two existing variations of the Multigrid technique that have been considered suitable for parallel computation and describes a new algorithm which is designed to exploit a greater level of concurrency than the previous schemes. Based on architectures proposed in the literature, a series of interprocessor communication models is developed to serve as a basis for comparing the selected algorithms. The algorithms are tested experimentally and their performance for each communication model is illustrated.

This research was supported by the National Aeronautics and Space Administration under NASA contracts no. NAS1-15810 and no. NAS1-16394 while the authors were in residence at ICASE, NASA Langley Research Center, Hampton Va. 23665. Addional support for the first author was provided by NSF Grant MCS-8108512.

Highly Parallel Multigrid Solvers for Elliptic PDEs: An Experimental Analysis.

Dennis Gannon [1]

Department of Computer Sciences, Purdue University

John Van Rosendale

ICASE, NASA Langley Research Center

1. INTRODUCTION

Most recent research in parallel algorithms for solving elliptic partial differential equations has enphasized direct methods. This work ranges in its degree of parallelism from the systolic band solvers of Kung and Leiserson [18] to a full nested dissection elimination (George and Liu [21] and Gannon [10]) and the fast Poisson solvers of Sameh, Chen and Kuck [25]. In the case of an elliptic partial differential equation on a square domain that has been discretized as a m by m mesh, the band solver requires $O(m^2)$ processors and m^2 parallel time steps. The nested dissection method requires the same number of processors but solves the problem in cm steps where c is a rather large constant. If the problem is the Poisson equation, the Sameh-Kuck scheme requires only $c * log(m)$ steps where, in this case, c is a small constant.

To devise a scheme that is more general than the fast poisson solver, but retains sublinear time bounds it seems necessary to turn to iterative algorithms. While there is a considerable body of work devoted to relaxation

[1] Work supported by NSF grant MCS-8109512 and ICASE, NASA Langley Research Center.

methods for vector processors, research on highly parallel iterative algorithms has been limited. Variations on the S.O.R. and Conjugate Gradient methods have been studied by Adams [1] for the mesh-connected Finite Element Machine. Baudet [5] considered chaotic relaxation algorithms, which can be performed on parallel computers. Brandt [7] has considered a parallel version of the multigrid method for which it can be shown that solving an elliptic problem to within a constant multiple of the truncation error requires no more than $O(log^2(m))$ time steps on m^2 processors.

In this paper, we extend Brandt's work by analyzing the performance of several multigrid algorithms suitable for parallel architectures. The plan of the paper is as follows. In section 2 we describe two algorithms derived directly from their standard serial formulations and give a simple analysis of the resulting parallel computational complexity.

In section 3 we show that the multigrid family may be extended to include an algorithm well suited to very high levels of concurrency. Each of the three algorithms considered here is easily described in terms of a small set of array valued parallel operators. To evaluate the performance of these methods we analyze the behavior of this set of operators in terms of the time complexity with respect to various inter-processor communication models. This analysis is given in section 4 for several classes of proposed VLSI and Multi-Microprocessor system architectures. In section 5, the communication cost models are combined with a set of numerical experiments to yield an approximate profile of the behavior of this family of algorithms over a wide range of highly parallel systems.

Section 6 considers the case of limited processor systems, and considers the extrapolation of experimental results to three dimensional problems.

2. MULTIGRID ALGORITHMS.

Of the wide variety of multigrid algorithms considered in the literature, two seem likely to perform best on parallel computers. The first of these is an algorithm related to the type of method considered in most theoretical research. The second is one type of multigrid algorithm commonly used in practice.

Multigrid algorithms are equally applicable to finite element or finite difference discretizations of elliptic boundary value problems. Suppose, for example, we wish to solve the linear system created by a discretization on a rectangular grid. To solve this linear system, multigrid algorithms employ two basic operations - relaxation iterations on this grid, and solution of related problems on a coarse grid. If the original grid has mesh spacing h, a coarse grid with mesh spacing $2h$ is usually used. An approximate solution of the given elliptic boundary value problem is often obtained on this coarser grid to provide a good starting value for iteration on the original fine grid. But, more importantly, solution of related problems on this coarser grid can be used to accelerate the convergence of the iteration on the original fine grid. Solving these related coarse grid problems efficiently reduces the long wavelength error components on that grid. This leaves only error components on the fine grid having wavelenghts comparable to the mesh spacing, and these can easily be removed by point iterative methods.

Fedorenko [9] observed the related problems on the coarser grid are again discretized elliptic boundary value problems and can be solved by the same techniques. This leads to a recursive algorithm where the solution of the problem on the original grid having mesh size h requires solution of a sequence of problems on the coarser grid with mesh size $2h$. Each of these coarse grid problems can be solved using an iteration periodically accelerated by solution of problems on even coarser grids having mesh size $4h$. The recursion is carried to a depth where one encounters a grid so coarse that either direct or iterative methods have trivial computational cost.

To describe such algorithms more precisely, suppose we have a family of finite element spaces $\{M_i\}_{i=0}^{n}$, nested in the sense that

$$M_{i-1} \subset M_i, \quad 1 \le i \le n,$$

and suppose the elements of space M_i have size approximately 2^{-i}. We will let superscripts denote membership in the corresponding space, so that u^i will be a function in M_i. Let L_i be the finite element operator approximating the elliptic operator on space M_i. Also, at each level, let P be a parameter controlling the number of recursive calls made to the next level, let J be a real parameter whose ceiling is the number of smoothing iterations performed at each level and let r be the rate at which J changes from level to level. The algorithm can then be described by the pseudo Pascal procedure below.

```
Procedure MG( var $u^i$, $f^i$: function; i,P: integer; r,J: real);
  var $v^{i-1}$, $g^{i-1}$: function;
      j: integer;

  begin
    if i = 0 then
       solve $L_i u^i = f^i$ by Gaussian elimination
    else begin
      perform  $J$  Jacobi iterations to approximately solve $L_i u^i = f^i$;
      $v^{i-1}$ := 0;
        (* generate a residual for a coarser gird problem *)
      $g^{i-1}$ := $\Pi_{i-1}(f^i - L_i u^i)$;
      for j := 1 to P do
              MG($v^{i-1}$, $g^{i-1}$, i-1, P, r, J*r);
      $u^i$ := $u^i + \Psi_i(v^{i-1})$;
    end;
  end;
```

Here Π_{i-1} and Ψ_i are projection and injection mappings used to accomplish the change of bases required in going between M_i and M_{i-1}. Mathematically,

$$\Psi_i : M_{i-1} \dashrightarrow M_i$$

is just the natural injection of M_{i-1} into M_i written in terms of the finite element basis for these spaces. The projection

$$\Pi_i : M_i \dashrightarrow M_{i-1}$$

is the adjoint of Ψ_i, thought of as a linear transformation between finite dimensional vector spaces.

Procedure MG can be used to solve elliptic boundary value problems with an optimal complexity bound. That is if N unknowns are needed to represent a solution in the finest finite element space M_n, one can obtain a solution which is accurate to within truncation error of the finite element equations for M_n in $O(N)$ serial operations. To obtain this optimal bound, one uses procedure MG first to generate a good solution in the coarsest space M_0. Then using this solution in M_0 as a starting value, one uses MG to generate a good solution in M_1, and so on. If instead, one applied procedure MG directly on the first space M_n, without first obtaining a good starting value from the coarser grids, an $O(N*log(N))$ bound would result.

Though the method used to obtain a good starting value for multigrid iteration in M_n is important, our main focus will be on the computational complexity and convergence rate of the iteration in M_n. To this end, we define a procedure Outer, to carry out this iteration.

```
Procedure Outer(var $u^n$, $f^n$: function; n,P,k: integer; r,J: real);
  var j: integer;

  begin
     for j:= 1 to k do
          MG($u^n$, $f^n$,n,P,r,J);
  end;
```

A number of multigrid algorithms considered in the literature are equivalent to various choices of the calling parameter for Outer. We look briefly at three possible choices of these parameters:

1. $P = 2, \ \tau = 1$
2. $P = 1, \ \tau > 2$
3. $P = 1, \ \tau = 1$

P controls the number of recursive calls procedure MG makes to itself in order to solve the related coarse grid problems and τ controls the relative number of inner iteration performed in these recursive calls. Thus, either r>1 or P>1 causes more inner iterations to be performed on the coarse grids. The first choice is equivalent to the type of multigrid algorithm considered in most theoretical work on multigrid methods. In particular, such algorithms are examined in Fedorenko [9], Hackbusch [14], Nicolaides [24], and Bank and Dupont [2]. The second choice is a variant described in Van Rosendale [30]. The third choice corresponds to one of the types of algorithms considered by Brandt [8], McCormick [23] and others.

For all three choices of the calling parameters, it is possible to prove the the multigrid iteration in procedure Outer has a spectral radius $\rho < 1$ independent of the number of levels n, i.e. the convergence rate is independent of the mesh spacing $h = 2^{-n}$. Let $\{u_j\}_{j=0}^{k}$ be the sequence of approximate solutions occurring in procedure Outer. That is, let u_0 be the approximate solution in M_n before the first call to *MG*, and let u_1 be the approximate solution before the second call. Also let $\bar{u}$ be the true discrete solution, and suppose the elliptic boundary value problem is second order, self adjoint and $H^{1+\alpha}$ regular for $\alpha < 0$. Then one can show

Theorem 2.1: Let P = 2 and r = 1 or let P = 1 and fix r > 1. Then for any $\sigma>0$ there exists $k = k(P, r, \sigma)$, such that

$$\|u_k - \bar{u}\| < \sigma\|u_0 - \bar{u}\|$$

The norm here is the natural energy norm but the same result holds for the L_2 norm on problems that are H^2 regular. In either case, the spectral radius satisfies $\rho \leq \sigma$, since M_n is finite dimensional.

The proof of theorem 2.1 is given in Bank and Dupont [2] for the case P = 2, r = 1, and in Van Rosendale [30] for the other case. There is an error in the extension to locally refined grids in Van Rosendale [30], but for quasi-uniform grids, such as those considered here, there is no difficulty.

For the third choice of parameters considered, P = 1, r = 1, an analog of Theorem 2.1 has been proven recently by Braess[6], Hackbusch[15] and McCormick[23]. Thus it is reasonable to assume that the iteration in procedure Outer converges about equally fast under each of the three choices of the calling parameters being considered. This implies that, up to a constant factor, the serial cost of reducing the error by a fixed amount will be the same for all three methods.

On a parallel computer, the situation is quite different. The first choice of parameters, P=2, r=1, will perform quite poorly, since the recursion involved amounts to a binary tree traversal. The second choice, P=1, r>1, should be better and the third choice, P=1,r=1, should be the best, since the least work is done on coarse grids with this choice.

Calculating the computational cost for each of the multigrid iterations in procedure Outer bears out these conclusions. Assume we have a total of $N=m^2$ processors that may operate in parallel. The operators L, Π, Ψ are computed as sparse matrix multiplications. For example, let M_b, $b=log(m)$ be defined in terms of the standard bilinear local finite element basis, $\{\varphi_{ij}\}_{i,j=1,m}$, on an m by m rectangular mesh where the basis function φ_{ij} is non-zero only on the node with coordinates (i,j). If a function u is written

$$u = \sum_{i,j=1}^{n} u_{ij}\varphi_{ij},$$

then the coefficient of $L_b(u)$ at node (k,l) is

$$L_b(u)_{kl} = \sum_{ij} L_{kl,ij} u_{ij}.$$

where the coefficient $L_{kl,ij}$ is non-zero only for the nodes neighboring the node at (k,l). Consequently, if one assigns one node to each processor and ignores (for now) the communication delay in transporting the values u_{ij} from node (i,j) to each of its neighbors, the time to compute $L(u)$ is independent of N. Similarly, Ψ_{b+1} is computed at each node as local averages of values from neighbor nodes. The injection is accomplished by first mapping u_{ij} to the odd indexed nodes of M_{b+1}, i.e. let

$$\begin{aligned} u'_{ij} &= u_{\frac{i-1}{2},\frac{j-1}{2}} \quad \text{for } i \text{ and } j \textit{ odd} \\ &= 0 \qquad \textit{otherwise}. \end{aligned}$$

The coefficients of Ψ_{b+1} are given by

$$\Psi_{b+1}(u)_{kl} = \sum_{s,t=-1}^{1} (1/2)^{|s|+|t|} u'_{k+s,\,l+t}.$$

The adjoint of Ψ_{b+1}, Π_b is expressed as

$$\Pi_b(u)_{kl} = \sum_{s,t=-1}^{1} (1/2)^{|s|+|t|} u_{2k-1+s\,,\,2l-1+t}$$

where k and l each range from 1 to $\frac{m}{2}$. Again ignoring communication costs, the parallel time for both Ψ and Π is independent of N, and is roughly equal to that required to compute $L(u)$. If we set the time for one of these operations as unity, then the total time required per inner iteration in procedure Outer is:

Parallel Time for One Outer Iteration

case	P	r	*Time*
1	2	1	$c\,(J+2)N^{1/2}$
2	1	$1<r<2$	$c\,(J+2)N^{\log_4 r}$
3	1	1	$c\,(J+2)\log_4 N$

Here c is a constant near one and J is the number of Jacobi inner iterations done on the finest space M_n. N is the number of nodes or variables of M_n.

To solve the elliptic boundary value problem to within truncation error, one could first use procedure Outer with some fixed value of k on the coarsest space M_0, then use the result as a starting value for Outer applied to M_1, and so on. This would yield a good solution in the finest space M_n. The computation time would be the same as given above but multiplied by a factor of $\log_4 N$. Thus we have:

Parallel time for a Complete Solution	
case	*time*
1	$ck(J+2)(log_4 N)N^{1/2}$
2	$ck(J+2)(log_4 N)N^{\log_4 N}$
3	$ck(J+2)(log_4 N)^2$

3. CONCURRENT ITERATION

Though the estimated parallel solution time for the algorithms just considered look reasonably good, one can ask whether these algorithms exploit parallel architecture as fully as possible. Suppose, for example, one has a number of processors comparable to the number of fine grid mesh points. Then when iterating or interpolating on the finer grids, processors will be well utilized. However, when iterating or interpolating on the coarsest grids, processor utilization would be quite poor. In this section a new algorithm is described, which is intended to utilize parallel architecture more fully than the algorithms in the last section.

The effectiveness of multigrid iterative algorithms can be viewed in a number of ways. One way of looking at it is to note that point iterative methods, such as S.O.R. are effective at reducing Fourier error components having wavelengths comparable to the mesh spacing, but do poorly on error components having much longer wavelengths. Multigrid iteration is effective, since it reduces such long wavelength components on coarse grids, where the mesh spacing is comparable to their wavelength.

In this view, one is using the projection operators in multigrid algorithms to decompose residuals into their short and long wavelength Fourier components. Pushing this idea slightly further, one can use these projections to decompose a function on the finest grid into components on every grid. In this way one can obtain a crude analog of the Fourier decomposition of a fine grid function, which we will call an "approximate spectral decomposition."

To see how this would work, let q^n be a given function in the finest finite element space, M_n, and let $p^i \in M_i$, $0 \le i \le n$, be a family of functions, one in every grid level. Then one can consider the following sequence of operations:

```
    $p^n := q^n$;
    for i := n downto 1 do begin
1.         $p^{i-1} := \Pi_{i-1} p^i$;
2.         $p^i := p^i - \Psi_i p^{i-1}$;
    end; (* for loop *)
```

Here step 1 in the loop sets p^{i-1} to the smooth part of p^i. Because the interpolation Ψ_i is the natural injection between finite element spaces, step 2 reduces p^i, so that $p^i + p^{i-1}$ is conserved on each pass through the loop.

On completion of these operations, one has

$$q^n = \sum_{i=0}^{n} p^i .$$

Here p^0 will be the smoothest part of q^n, and p^n the most oscillatory. The functions p^i, $0 \le i \le n$, are increasingly oscillatory with increasing i.

A simple multigrid algorithm can be based on these ideas. In order to solve $L_n u^n = f^n$, one could carry out the following sequence of operations:

1. Form an approximate spectral decomposition of f^n :

$$f^n = \sum_{i=0}^{n} g^i$$

2. Approximately solve the problems:

$$L_i v^i = g^i, \quad 0 \le i \le n,$$

by point iterative methods.

3. Sum the solutions on all grid levels:

$$\bar{u}^n = \sum_{i=0}^{n} v^n$$

The point iterative methods in step 2 here would converge rapidly for the first few iterations on each grid level, but would then slow down. This rapid initial convergence occurs, since the approximate spectral decomposition in step 1 makes each of the data functions g^i one of the most oscillatory functions in its finite element space M_n. However, after a few iterations, the residual on each level would be relatively smooth, and the convergence correspondingly slow.

To avoid slowly convergent iterations in step 2, one could perform only a few iterations on each level, and then proceed to step 3. The resulting approximate solution $\bar{u}^n$ might not be very accurate, but could be iteratively improved by repeating the algorithm using the residual as data. Such iterative improvement would be necessary in any case, since the iterations in step 2 invert a different discrete operator on each grid level. Each of the operators $\{L_i\}_{i=0}^{n-1}$ approximates the fine grid discrete operator L_n , but for small i this approximation can be quite poor.

The algorithm just described would be reasonable on a serial computer, but is not as parallel as we would like, for the architectures to be considered. The iteration in step 2 of this algorithm can be done concurrently on all levels, but the interpolations in steps 1 and 3 must be done level by level. To avoid this bottle neck, we look at a modified algorithm in which the amount of interpolation needed is minimized. The algorithm is a two step iteration which may be sketched as follows:

```
    for i := 1 to itmx do
    begin
1.      perform J smoothing inner iterations on each grid level
2.      interpolate solutions and residuals between adjacent levels
    end
```

The iterations in step 1 here can be done in parallel on all grid levels. The same is true for the interpolations in step 2, hence the name "concurrent iteration." Figure 3.1 shows the data paths in this algorithm.

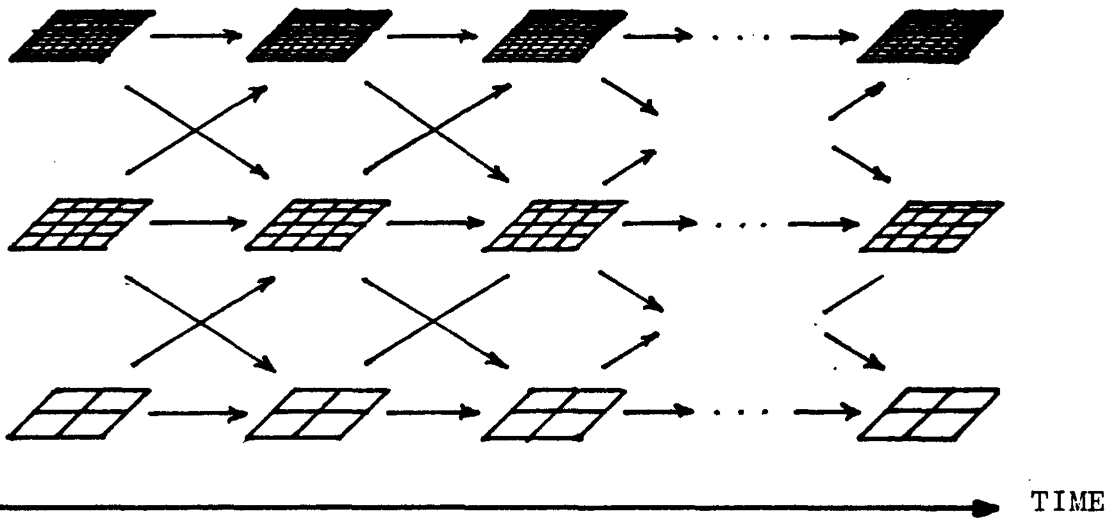

Figure 3.1 Grid Level Data Flow of Concurrent Iteration.

Achieving data paths such as these, which can easily be mapped onto parallel architecture, is the underlying motivation for this algorithm.

Mathematically the algorithm required is somewhat more complex than those in the last section. First we need a procedure to shift the data in a family of functions, $\{p^i\}_{i=0}^{n}$ one level, by performing a sequence of injections. Defining a data type "vect_fn" representing such a family of functions, this procedure, *minj*, is given below.

```
procedure minj (var $\bar{p}$ : vect_fn);
var i : integer; $\bar{q}$ : vect_fn;
 begin
    for i := 1 to n pardo
        $q^i := \Psi_i p^{i-1}$;
    for i := 1 to n pardo
        case i of {
           0 : $p^i$:=0;
           n : $p^i := p^i + q^i$;
           otherwise : $p^i := q^i$ ;
        }
 end;  (* of minj *)
```

In this procedure, both the loops explicitly described, and the loops hidden in the interpolations $\{\Psi^i\}_{i=1}^{n}$ can be done entirely in parallel. The computation may be viewed as a vector of array operators. Also note that this procedure conserves $\bar{p}$ in the sense that the sum $\sum_{i=0}^{n} \Psi_n .. \Psi_{i+1} p^i$ remains unaltered. Because the operator Ψ is the natural injection of a subspace of a vector space, the operator $\sum_{i=0}^{n} \Psi_n \cdots \Psi_{i+1} z_i$ is nothing more than the pointwise sum of $\{z_i\}_{i=0}^{n}$ viewed as functions over the defining domain. In the following paragraphs we abbreviate this operation as $\sum_{i=0}^{n} z_i$.

A procedure similar to *minj* can be given to perform projections between grid levels. This operator, *mprj* is described below.

```
procedure mprj (var $\bar{p}$ : vect_fn);
var i : integer; $\bar{q}$ : vect_fn;
begin
   for i := 0 to n-1 pardo
      $q^i := \Pi_i p^{i+1}$ ;
   for i := 0 to n pardo
      case i of {
         0: $p^i := p^i + q^i$
         n: $p^i := p^i - \Psi_{i-1} q^{i-1}$
         otherwise: $p^i := p^i + q^i - \Psi_{i-1} q^{i-1}$
      }
end;     (* of mprj *)
```

This procedure can also be done entirely in parallel and conserves the sum $\sum_{i=0}^{n} p^i$. Also note that n successive calls to this procedure would perform the approximate spectral decomposition of p discussed at the beginning of this section.

The two procedures just described, *mprj* and *minj*, are the basis for the concurrent iteration algorithm. Suppose we also have a procedure $jacobi(v^j, q^j)$, which approximately solves

$$L_j v^j = q^j$$

by performing a fixed number of Jacobi smoothing iterations. Then the concurrent iteration algorithm for the discrete problem

$$L_n u^n = f^n,$$

is as follows.

```
procedure concurrent (var $u^n$, $f^n$ : function; J, itmx : integer);
(* performs itmx concurrent multigrid iterations to
  approximately solve $L_n u^n = f^n$        *)

  var $\bar{v}$, $\bar{d}$, $\bar{q}$ : vect_fn;
      i,j : integer;

begin
   for j := 0 to n pardo begin          (* initialization *)
      $v^j$ := 0;
      $q^j$ := 0;
   end;

1. $q^n$ := $f^n$;                        (* put data in $\bar{q}$ *)
   for i := 1 to itmx do begin            (* outer iteration loop *)
        for j := 0 to J pardo
2.          jacobi ($v^j$, $q^j$);        (* performs inner relaxations *)

        for j := 0 to n pardo begin
3.          $d^j$ := $L_j v^j$;           (* compute data corresponding to $v^j$ *)
4.          $q^j$ := $q^j - d^j$;         (* compute residual *)
        end

5.      mprj($\bar{q}$);                  (* shift $\bar{q}$ to coarser grids *)
6.      minj($\bar{v}$);                  (* shift $\bar{v}$ to finer grids *)
7.      minj($\bar{p}$);                  (* shift $\bar{p}$ to finer grids *)

        for j := 0 to n pardo
            $q^j$ := $q^j + d^j$;         (* put all data in $\bar{d}$ *)
   end;            (* of outer iteration loop *)

  $u^n$ := $v^n$;
end;                  (* of procedure concurrent *)
```

The intuitive meaning of the steps here is as follows. In step 1, the functions $\{q^j\}_{j=0}^n$ contain the data and satisfy

$$f^n = \sum_{j=0}^n q^j$$

In step 2, procedure *jacobi* performs a few relaxation iterations on each grid to approximately solve the problems

$$L_j v^j = q^j, \qquad 0 \le j \le n.$$

Thereafter, the data d^j corresponding to v^j and residual q^j on each level are computed in steps 3 and 4. At the end of step 4 data is conserved in the sense that

$$\sum_{j=0}^n (q^j + d^j) = f^n$$

holds up to roundoff error.

The heart of the algorithm is in steps 5, 6 and 7. In step 5, residual data in $\bar{q}$ is shifted to coarser grids to speed convergence of procedure jacobi. Ideally, we would want $\bar{q}$ to be an approximate spectral decomposition of the residual,

$$\sum_{j=0}^{n} q^j,$$

with each q^j in the null space of the projection Π: $M_j \to M_{j-1}$. This would mean each q^j was one of the most oscillatory functions in M^j, so subsequent Jacobi iterations woud converge rapidly; however, in order to maximize parallel speed, *mprg* is only called once here instead of the n-1 times required to complete the approximate spectral decomposition.

Steps 6 and 7 shift the approximate solutions $\{v^j\}_{j=0}^{n}$ and corresponding data $\{d^j\}_{j=0}^{n}$ to finer grids. The idea here is to shift the approximate solution functions $\{v^j\}_{j=0}^{n}$ to the finest grid, where v^n will eventually converge to the true discrete solution. Equally important, the corresponding data must be simultaneously shifted or procedure jacobi will go on solving for it, leading to divergence of the outer iteration. By shifting an approximate solution v^j and corresponding data d^j to the next finer grid level, very little new residual will be created. To be precise, this newly created residual will be

$$r^{j+1} = L_{j+1}\Psi_{j+1}v^j - \Psi_{j+1}d^j$$

This residual r^{j+1}, which will be small if L_j and L_{j+1} approximate each other well, will percolate between grid levels just as the original data f^n did, and will eventually be solved for as well.

One could give more algebraic explanations for this algorithm, but since no convergence results have been proven, we content ourselves with this heuristic description. The effectiveness of this algorithm is demonstrated by our numerical tests. If this algorithm becomes of practical value, one might wish to search further for theoretical convergence results.

4. ARCHITECTURAL CONSIDERATIONS

In this section we consider the complexity issues associated with the algorithms of the previous section when programmed on a variety of highly parallel architectures. More specifically, we give a rough classification of hardware systems in terms of the relative costs of data movement and arithmetic steps required for the basic operations L, Ψ, and Π in the multi-grid family of algorithms. Each of the systems described here will be assumed to have a very large number of relatively simple processors, each having a local memory (or a segment of a shared memory) large enough to hold its own program and the data associated with at least one node of the grid (the real values f, u, v, d, q from the previous section). The basic distinction made between the architectures will be in terms of the structure of the inter-processor communication networks.

Given an elliptic p.d.e. defined on a domain which can be mapped into a square mesh of dimension m by m where $m = 2^n + 1$, an ideal parallel architecture for the multi-grid algorithms described above would be a network of processors configured into n levels where the i^{th} level is a square grid of edge dimension $2^i + 1$. The interconnection structure is illustrated in figure 4.1. The basic operations of L, Ψ and Π are easily accommodated on this structure. The Jacobi smoothing is defined in terms of the operation $L(u)$ which, when approximated with a 5 or 9 point template, involves communication only along the nearest neighbor connections in the mesh. As described in section 2, Injection, Ψ, and Projection, Π are defined by a nearest neighbor "averaging template", of the same complexity as the $L(u)$ operation, which is preceded (followed) by a mapping between the nodes of the coarse grid at level i and the odd indexed nodes of the finer grid at level $i+1$.

The network in figure 4.1 completely reflects the inter-process data flow of the multi-grid algorithms. Consequently, the problem of programming these

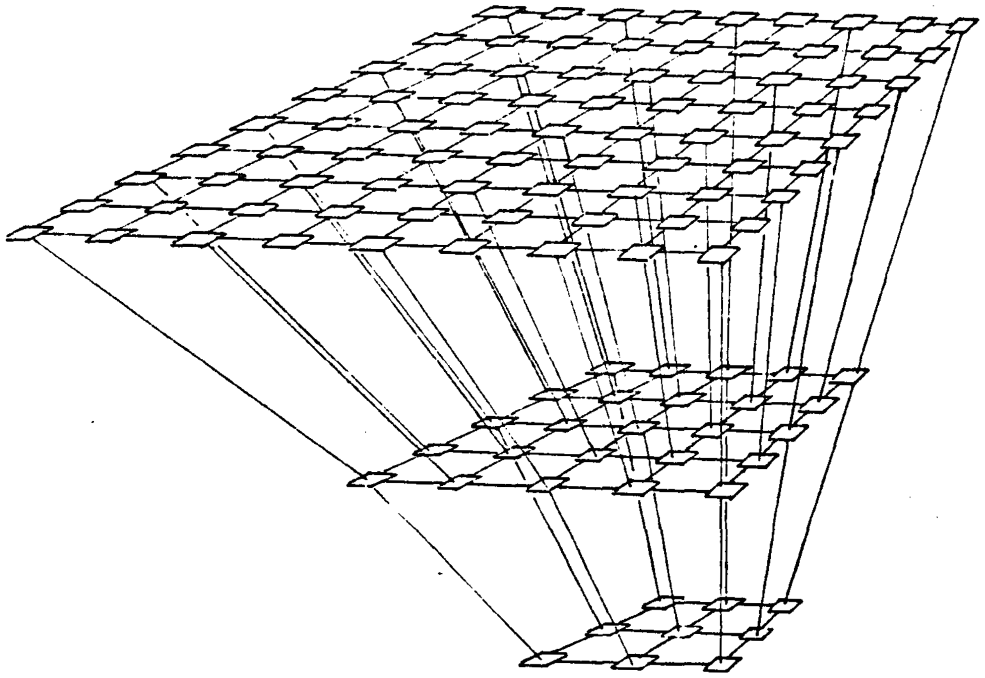

Figure 4.1 Basic Multi-Grid Processor Array.
In the horizontal planes, each processor is connected to its 8 nearest neighbors (only 4 neighbor connections are shown). Intergrid connections map processor (s,t) on level i to processor $(2s-1,2t-1)$ in level $i-1$.

methods on other systems reduces to the problem of emulating this network in the communication scheme of the architecture at hand. This task will be studied for four basic classes of highly parallel computer architectures.

Let R_j be the parallel time required to complete a Jacobi relaxation on grid j. Similarly, let I_j and P_j be, respectively, the complexity of Ψ and Π on the j^{th} grid. In general the quantity R_j will be a sum of three terms: the time required to complete the numerical computation carried out at each processor; the overhead time required by a processor to send and recieve messages from each of its neighbors; and the actual transit time required for message passing. The relative size of these terms will vary widely from system to system. The communication software overhead for an asynchroneous MIMD system typically involves complex buffering and handshaking protocols, and may dominate the arithimetic computation. For synchroneous, highly parallel VLSI based systems the communication overhead is very low but data transmission times through lenghty data paths or complex switch networks can be quite large. In general, the arithmetic and communication-software overhead terms in R_j will roughly equal the corresponding components in I_j and P_j. The primary difference between R_j and the latter pair is that Π and Ψ require the additional "inter grid" data movements.

Our approach to characterizing these differences will be to study four classes of highly parallel system designs and extract four analytical models for the added comunication cost of the "inter grid" transmissions.

The simplest cost model is one in which all communications between processors have same complexity, i.e.

$$I_j = P_j = R_j = R_1, \quad j=1,..,n \tag{4.1}$$

While this estimate will hold for for some model architectures such as figure 4.1, it does not hold in many of the interesting cases which represent machines either planed or under construction.

4.1 Mesh Connected Arrays.

Systems such as the Finite Element Machine (Storaasli [29] Jordan [16]) or the Massively Parallel Processor (MPP) (Batcher [4]) are configured as planar η by m nearest-neighbor connected grids of processors. Figure 4.2 shows 3 methods for mapping the many layered MG network into a planar mesh. 4.2a and 4.2b show embeddings that are suitable for the Concurrent Iteration algorithm. Because computation must proceed in parallel on all n grid levels, each level must be explicitly represented.

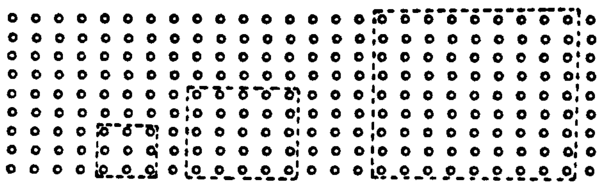
Figure 4.2a $\eta=3m-1$

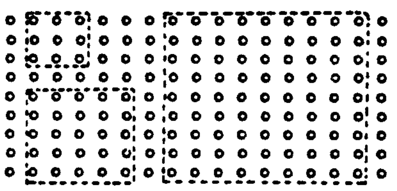
Figure 4.2b $\eta=\frac{3(m-1)}{2}+2$

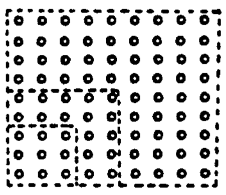
Figure 4.2c $\eta=m$

Of the two methods, 4.2b is clearly the most efficient packing of the multilevel structure ($\eta=\frac{3(m-1)}{2}+2$ versus $\eta=3m-1$), but 4.2a has advantages that are described in the next section. Figure 4.2c shows the natural embedding for the multigrid algorithms of section 2. In this case the separate grid levels are processed sequentially and therefore one may "overlay" them on the host grid. The shortcoming of all three structures lies with the emulation of the data paths connecting the grid levels together. Given an array of values associated with a grid at level j, the data movement required to map this to level $j+1$ can be decomposed into two "expand" operations. Assume that each processor may read from and write to any neighbor and that we have a chain of m processors such that processors 1 through $m/2$ each hold an item of data. An "expand" move is defined to be the sequence of steps required to get the processor at node $2i$ to hold the data item from node i from $1 \le i \le m/2$. The inverse operation is known as a "compress". The reader should have no trouble seeing that the processors in a nearest neighbor network can complete an expand or squeeze by a set of parallel "bucket brigade" write-read steps in time $m/2$.

By applying the expand along each row in parallel and then along each column in parallel the data item in position (i,j) is moved first to (2i-1,j) then to (2i-1,2j-1). When applied to the embedding 4.2a all subgrids are expanded in parallel in 2m-2 steps. For embedding 4.2c, the i^{th} subgrid is expanded in 2^i-2 steps. (Embedding 4.2b requires a more elaborate algorithm, but can be completed in 3m/2 -2 steps).

Clearly, embeddings 4.2a and 4.2b permit simultaneous relaxation on all grids and, because relaxation is independent of the size of the grid, one has R_j = R_1 for $j=1..n$. Taking the communication algorithm described above into account, one obtains

$$I_j = P_j = R_1 + c_1 2^j \quad \text{for } \textit{embedding} \text{ 4.2c.} \tag{4.2}$$

$$I_j = P_j = R_1 + c_1 m \quad \text{for 4.2}a \text{ and 4.2}b \tag{4.3}$$

where c_1 is a small constant that relates the time required for a read and a write operation to R_1.

4.2 Mesh-Shuffle Connected Systems.

It has been proposed by Grosch [13] and Brandt [6] that, because of the awkward nature of the intergrid communication algorithm described above, the communication network of the mesh system be enhanced by the addition of a shuffle network in each row and column. The shuffle connection, shown in figure 4.3 for one row of 8 processors,

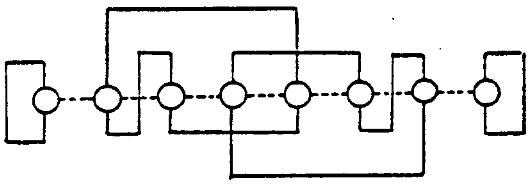

Figure 4.3 Shuffle connection on 8 nodes.

provides the exact connections that are simulated by the expand or compress described in 4.1. In particular, processor i is directly connected to processor $2i-1$ for $i \le \frac{n}{2}$. Embeddings 4.2a and 4.2c have the property that level j is mapped to level $j+1$. The expansion via the shuffle permutation is illustrated for a row of processors from figure 4.2a in figure 4.4.

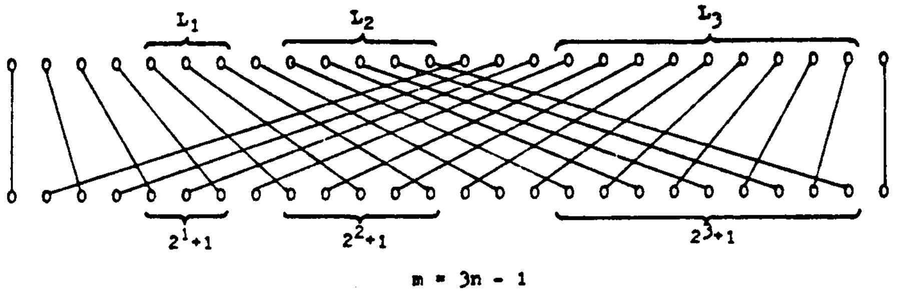

Figure 4.4 Expansion of three levels from one row of embedding 4.2a. Levels L_1 and L_2 are directly connected into the odd subsequences of levels L_2 and L_3 respectively.

The complexity of this operation is dependent upon the effect of propogation delay along the long wires in the network. In general, the longest line between level i and level $i+1$ will grow as 2^i. Consequently the cost model described by equations 4.2 or 4.3 will apply. If we assume that signal propogation is negligible then model 4.1 is applicable.

4.3 Permutation Networks.

A number of highly parallel architectures have been proposed that use some form of general permutation network for processor to memory or processor to processor communication. These include the Flow Model Processor of Burroughs [22], the PASM system [27], Ultracomputer [26] and the TRAC system [17]. The connection networks used in these systems can each be viewed as a network of $log(m)$ stages of m switches per stage. The family of data movements that can be simultaneously executed on such a network is usually a subset of the full permutation group. The Ω^{-1} network of Lawrie [19] (shown in figure 4.5 for $m=8$ processors) can be shown to admit a set

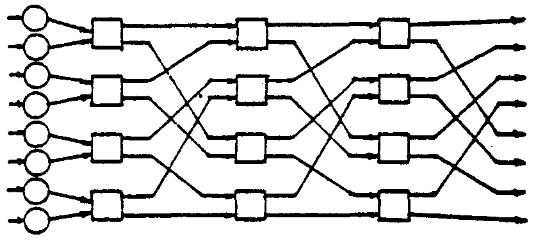

Figure 4.5. Ω^{-1} network for 8 processors.

of permutations that include uniform shifts and the compress operation described above [11]. Its inverse, the Ω network, can execute uniform shifts and the expand operation. Because ALL communications must pass through $log(m)$ switch stages, the cost model is equivalent to the constant delay model in equation (4.1) where the constant depends on the size of the largest grid.

To accelerate the communications on an individual grid level, it is possible to augment the Ω^{-1} network with a mesh connection as diagramed in figure 4.6. It is shown in [11] that this structure is well suited for a class of locally refined

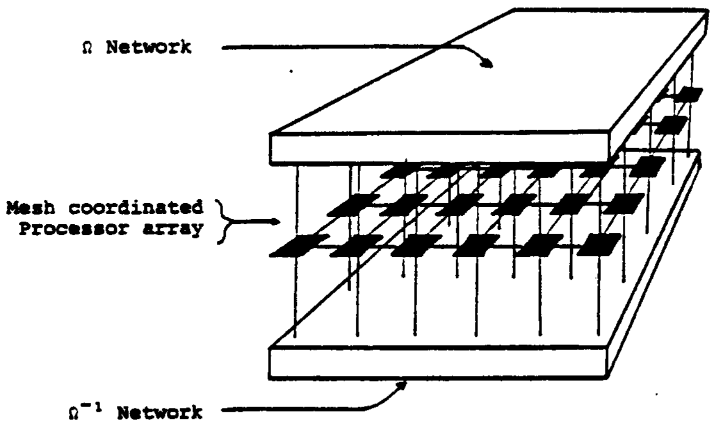

Figure 4.6. Mesh-Ω system.

grids as well as the uniform grids studied here. In this case, the mesh communications are uniform but the intergrid messages must traverse $n = log(m)$ switch stages. The appropriate model for both standard multigrid and Concurrent Iteration is given by

$$P_i = I_i = R_n + cn \tag{4.4}$$

4.4 Direct VLSI Embeddings.

All of the preceding architectures may be viewed as directly embedable in silicon for VLSI implementation. Unfortunately, they do not all possess properties considered desirable by current design criteria. In particular, the surface area of a chip is reguarded as an important resource. Designers are encouraged to provide area optimal embeddings of any network. A disadvantage of the shuffle exchange graph is that it may be shown to require area of order $O(\frac{m^2}{\log^2(m)})$ [20] for m nodes. The mesh-embeddings in figure 4.2 require area that is linear in the number of nodes, but the intergrid algorithm is complex. Figure 4.7 illustrates a linear area embedding of the complete multigrid network (figure 4.1). This structure is directly programmable on the proposed CHiP architecture [26] and has the further advantage that all connections are

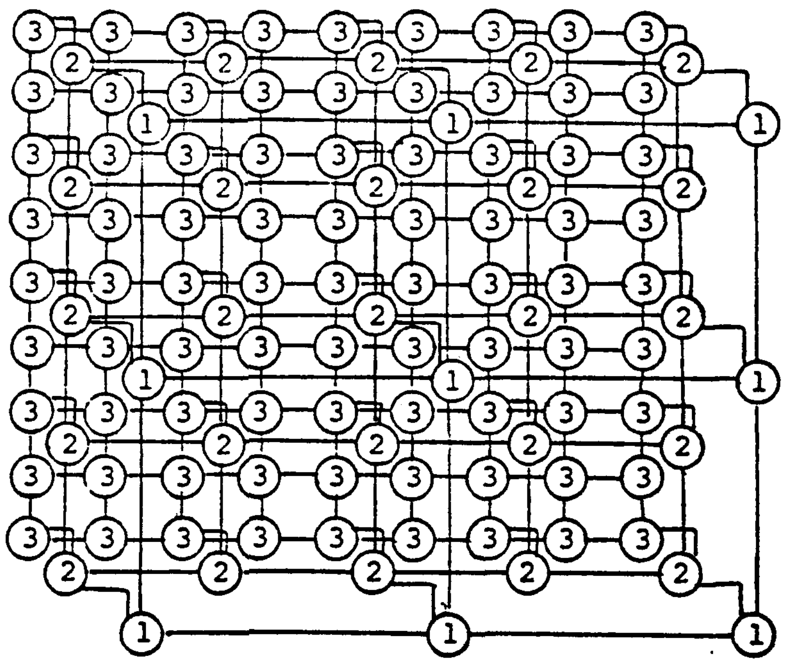

Figure 4.7. CHiP Processor Embedding.

completely implemented, which implies that the communication algorithm is simple. Unfortunately this method has one major drawback. The length of the data paths for the coarse grids grow exponentially with the maximum level of refinement. If one assumes that the propagation delay is linear in the length of the data path, then the appropriate cost relation is given by

$$P_i = I_i = R_i = R_n + c2^{n-i} \tag{4.5}$$

As with the shuffle network of section 4.2 one may wish to assume that c is small enough to be negligible.

The ideal embedding of figure 4.1 into silicoln would be one that has area that is linear in the number of nodes but has an upper bound (independent of n) on the length of the longest edge between two processors. Unfortunately, this pair of requirements for 4.1 can be shown to be unachievable.

5. Numerical Experiments

Theoretical computational complexity bounds have not been established for all of the multigrid algorithms considered here. Also, even when they are available, these bounds tend to be highly pessimistic. In this section we consider a sequence of numerical experiments designed to ascertain the practical value of the algorithms described above. This approach has the advantage that the impact of changes in the computer architecture on the choice of method is relatively easy to assess. In particular, the relative cost of data communication is found to have a major impact on the choice of algorithm.

The four communication models from section 4 are summarized in table 5.1 for a n level multigrid structure.

model	*relations*
constant	$P_j = I_j = R_j = R_n, \; j = 1..n$
log	$P_j = I_j = R_n + cn; \quad R_j = R_n$
linear	$P_j = I_j = R_n + c\,2^j; \quad R_j = R_n$
linear-VLSI	$P_j = I_j = R_j = R_n + c\,2^{(n-j)}$

Table 5.1. Communication cost models.

The constant c is hardware implementation dependent.

As indicated in section 4, the actual performance of any of the algorithms on a specific machine will depend on the particular design and hardware. In most cases the appropriate model is a linear combination of the constant model and two or three of the others, i.e.

$$P_j = I_j = R_j = R_n + c_1 2^j + c_2 n + c_3 2^{n-j}$$

for some constants c_1, c_2 and c_3. Rather than guessing at the values of these constants, an accurate profile of performance can be obtained by examining each component cost model separately. That is, each algorithm is tested against each of the cost models in table 5.1 with $c = 1$, and $R_n = 1$. For each algorithm and each model, these measurements yield an empirically derived performance function

$$F_{Algorithm,\,Model}: \textit{number of grid levels} \dashrightarrow \textit{execution time}.$$

Then, given the constants $c_1,\ldots,c_3$, and R_n for a particular hardware/software system, the performance of algorithm X as a function of problem size would be predicted from

$$(R_n - c_1 - c_2 - c_3)F_{X,const} + c_1 F_{X,log} + c_2 F_{X,linear} + c_3 F_{X,VLSI}.$$

Because our primary concern is the relative performance of the algorithms described in sections two and three under different parallel communication models, we have made relatively little attempt to "tune" the inner iterations. All methods were tested using the same Jacobi inner iteration with the same near optimal relaxation parameter. In a "production" code one would want to try Chebyshev acceleration and various other point relaxation techniques, such as red-black S.O.R. One could also consider inner iterations involving tridiagonal solvers, such as A.D.I. or Zebra relaxation, since there are fast parallel algorithms for band matrices. (Care must be exercised with the latter techniques because they substantially add to the communication complexity of the method and the fastest such algorithms introduce stability problems that must be dealt with.) We note, however, that any such optimizations are likely to improve all three methods tested below in an nearly equitable manner. (A slight edge in inprovement may go to the Concurrent Iteration algorithm because, as it will be shown, it depends most heavily on the spectral radius of the inner iteration).

Each algorithm was tested for each problem on grids of size m by m where $m = 2^n+1$ and n ranges from 3 to 6, i.ė. for algebraic systems of size 49, 225, 961, and 3969. In each test the number of inner iterations was held at $J=4$ (changes in the value of J modified the scale of the result but not the basic character.) The three methods tested are the concurrent iteration of section 3, the Log(m) time per Outer iteration scheme (P=1, r=1) of Brandt which shall be refered to as LO Multigrid (or LO-MG), and the root(m) time per Outer iteration method (P=1, r = 1.2) of section 2 which we shall call RO Multigrid (or RO-MG). The model problems studied here are all two dimensional scalar equations. Of the many features that should be tested to validate an iterative method for elliptic equations, we have selected three: the rate of convergence to a point below the truncation error of the discretization; the effect of domain shape; and the effect of a nonself adjoint operator.

5.1 Truncation Error. Let u be the exact solution to the differential problem and let u_∞^m be the exact solution to the discrete system defined on an m by m mesh on the unit square. If u_k^m is the approximate solution at outer iteration k, then given some constant c<<1, a reasonable measure of performance is the smallest value of k for which,

$$||u_k^m - u_\infty^m||_2 < c\,||u - u_\infty^m||_2 \qquad 5.1$$

In other words, when does the error in the discrete solution become dominated by the truncation error. Our first test is to consider this behavior for the simple equation

$$\nabla^2 u = -1/2\pi^2 sin(\frac{\pi}{2}x) sin(\frac{\pi}{2}y)$$

with boundary conditions $u=0$ on $x=0$ and $y=0$ and a vanishing normal derivative $\frac{\partial u}{\partial n}$ on $x=1$ and $y=1$. The exact solution is

$$u = sin(\frac{\pi}{2}x) sin(\frac{\pi}{2}y)$$

The test results are plotted in figure 5.1. Each graph plots performance functions $F_{method,\,model}$ for a given model and all three methods. The horizontal axis is problem size (measured in total number of grid levels) vs. parallel time to satisfy equation 5.1 when c was arbitrarily chosen to be 10^{-4}. Table 5.2 gives the detailed data for these plots. Also included are the mean spectral radius for each outer iteration and an approximate count of the serial complexity.

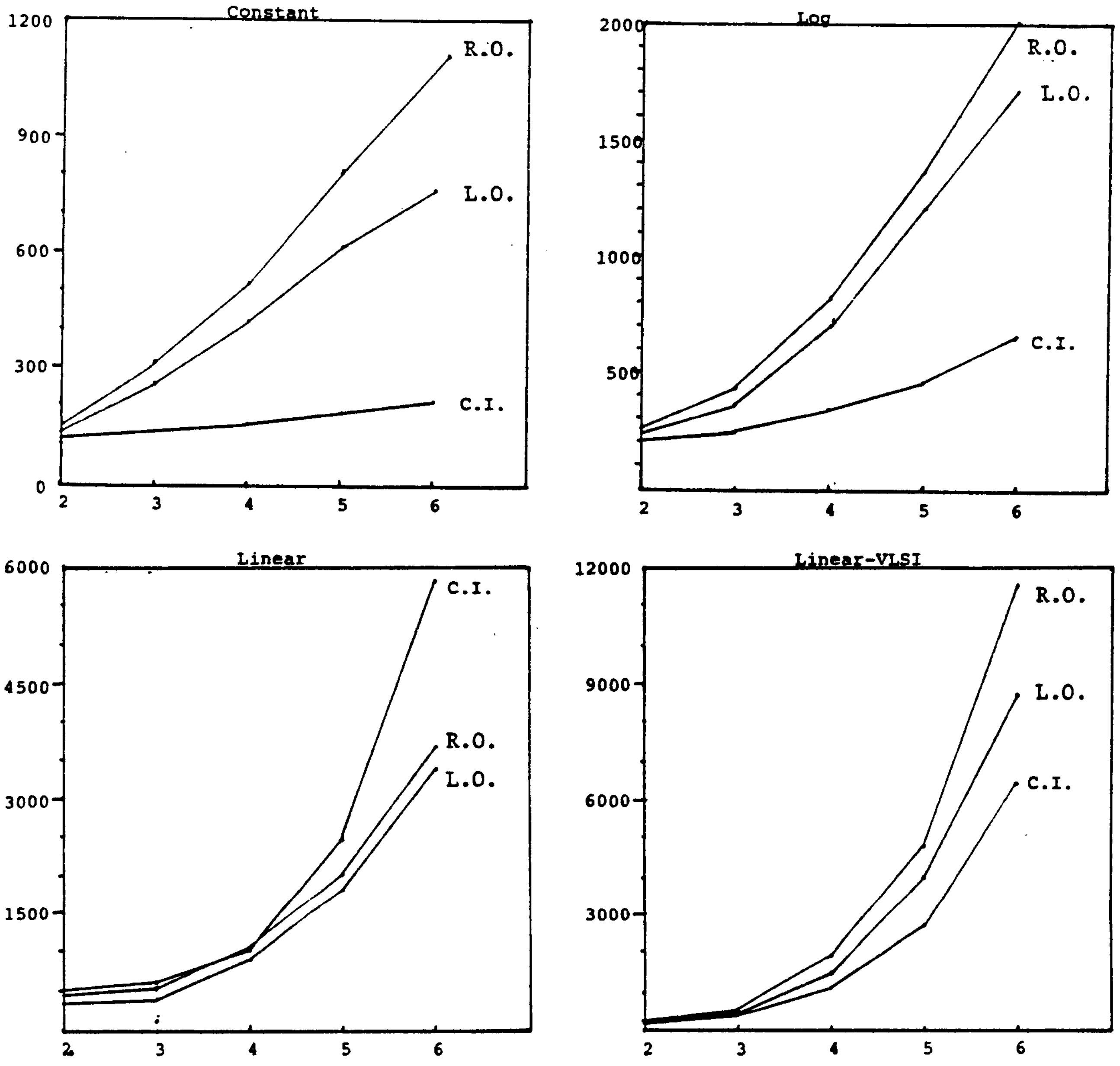

Figure 5.1. Execution time vs. grid size.

There are several points worthy of note in these results. First, unlike both the RO-MG and LO-MG algorithms, concurrent iteration does not have the property that its spectral radius is bounded independent of n. However for the constant and log cost communication models, the new algorithm exhibits enough concurrency to be substantially faster than the modified serial methods.

In the case of the linear cost model, a heavy toll is paid for fine grid interpolations. Both the concurrent iteration and the LO-MG scheme do one coarse grid interpolation for each outer iteration. As the number of grid levels increases the cost of fine grid interpolations begins to dominate the cycle time of each outer iteration. Hence, the method that requires fewer outer iterations, LO-MG, will be the fastest.

Table 5.2. Experiment 5.1 detailed results.

method	grid levels	spectral radius	constant	log	linear	serial	linear-VLSI
RO Multigrid	3	.34784	315	431	509	9036	692
	4	.34685	523	811	1045	42112	1869
	5	.34570	808	1368	2036	203296	4789
	6	.35086	1104	2004	3624	840948	11448
LO Multigrid	3	.37936	249	357	429	6540	576
	4	.38522	412	700	934	29404	1576
	5	.38286	610	1202	1920	127468	3968
	6	.37111	765	1715	3461	477300	8776
Concurrent Iteration	3	.37404	133	247	532	11305	532
	4	.43387	148	340	1108	50468	1184
	5	.46530	170	466	2464	232050	2720
	6	.51354	206	656	5876	1123966	6592

In the case of the linear-VLSI model the weight is on the coarse grid relaxations. The RO-MG algorithm is slowest because it makes the the largest number of coarse grid iterations. In this model the cost of inter grid communication is trivial and Concurrent iteration, exploiting more parallelism than LO-MG, is moderately faster (though, for the same reasons given in the linear model, one would expect RO-MG to dominate for larger problems.)

5.2 Domain Shape. A large number of fast solvers are suitable only for problems defined on the unit square. In the case of nonconvex domains many of these methods, such as the fast Poisson solver, break down. Ideally, a general iterative method should have its convergence rate largely unaffected by the existence of singularities at interior corners. To test this property consider the problem

$$\nabla^2 u = 1.0 \qquad 5.2$$

on the L-shaped domain in figure 5.2. The boundary conditions

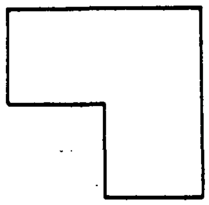

Figure 5.2. Domain for problem 5.2.

are that the normal derivative is zero on the outer boundary and the solution is constrained to zero near the reentrant corner. The experimental results are summarized in table 5.3 below. Because we do not have the exact solution to this P.D.E. we considered a different convergence criteria. The methods were run until machine precision was reached and the resulting solution, u_∞, was recorded. The solution process was repeated until the L_2 error satisfied

$$\| u_k^n - u_\infty^n \|_2 < 10^{-6}$$

where the constant 10^{-6} was chosen arbitrarily.

Table 5.3. Experiment 5.2 detailed results.

method	*grid levels*	*spectral radius*	*constant*	*log*	*linear*	*serial*	*linear-VLSI*
RO Multigrid	3	.33069	231	315	369	6396	512
	4	.33999	399	615	777	30000	1473
	5	.34069	596	993	1411	130556	3734
	6	.32832	880	1580	2696	582052	9704
LO Multigrid	3	.35928	213	305	365	5508	496
	4	.43962	362	614	812	25172	1400
	5	.51018	482	946	1464	93404	3232
	6	.56952	648	1448	2816	374952	7648
Concurrent Iteration	3	.61730	168	312	672	14280	672
	4	.65681	176	404	1316	60016	1408
	5	.67975	191	523	2764	260715	3056
	6	.69601	213	678	6072	1164193	6816

The basic performance profile of the three methods with respect to the four models is relatively unchanged. However, there is a change in spectral radius behavior. The RO-MG method remains bounded (this is provably the case), but the LO-MG method shows a substantial degeneration in convergence rate.

5.3 Non-self Adjoint Operators. One important use of fast elliptic solvers is as a component of an implicit scheme to solve the time dependent problem of the form

$$\frac{\partial u}{\partial t} = \frac{\partial u}{\partial x} + \alpha_n \nabla^2 u$$

where α_n is an artificial viscosity that is chosen so that on a grid with mesh spacing h_n then

$$\alpha_n = O(h_n)$$

At each time step the elliptic equation

$$\alpha_n \nabla^2 u + \frac{\partial u}{\partial x} = f \tag{5.3}$$

must be solved for some function f of u on the previous time step. For the third experiment we solve the family of problems defined by $\alpha_n = 2^{-n}$ and $f = 1.0$ on the domain D in figure 5.4 defined by removing the triangle $\frac{1}{2} > y > x$ from the unit square. This experiment corresponds to advancing the time dependant problem solution one step. The boundary condition is that $u = 0$ on ∂D.

Table 5.4. Experiment 5.3 detailed results.

method	*grid levels*	*spectral radius*	*constant*	*log*	*linear*	*serial*	*linear-VLSI*
RO Multigrid	3	.13211	294	402	474	8376	2572
	4	.17757	523	811	1045	42112	6510
	5	.20682	851	1443	2161	217844	15697
	6	.18955	1160	2110	3856	905672	35976
LO Multigrid	3	.18662	285	409	493	7572	2392
	4	.25718	512	872	1178	37868	5968
	5	.31128	770	1522	2490	170048	13824
	6	.31372	999	2249	4751	681996	30336
Concurrent Iteration	3	.21557	112	208	448	9520	448
	4	.46492	225	516	1680	76725	1800
	5	.62076	338	992	4868	461370	5408
	6	.68554	409	1294	11564	2233549	13088

In this case we considered the error in the sup norm: the results listed in table

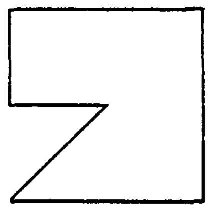

Figure 5.3. Domain for problem 3.

5.3 are the times required to reduce the L_∞ error

$$\|u_k^n - u_\infty^n\|_\infty \leq 10^{-12}.$$

Because the operator differed from the previous problems the inner iteration relaxation parameter was optimized to yield a reasonably good spectral radius when $n=3$. As the results indicate, the spectral radius of the concurrent iteration algorithm degenerates as n increases, but the parallel time for the log and constant models continues to be better than for the other schemes.

6. CONCLUSION

There are at least two major areas of inquiry left untreated by the preceding analysis. The first of these is algorithm performance in the case where the number of processors is limited. The concurrent iteration algorithm requires roughly $\frac{4}{3}m^2$ processors in its fully parallel form to solve a problem on a m by m grid. The LO-MG algorithm uses m^2 processors. Observe that the serial complexity for C.I. in experiment 5.1 with $m=65$ is 1123966, whereas LO-MG requires 477300. Based on a zero cost communication model these figures yield a processor utilization of 100% for C.I. but only 15% for the LO-MG scheme. This fact, together with an analysis of the algorithms, leads one to conclude that the execution time for C.I. will double if the number of processors is cut in half but LO-MG, not fully utilizing its m^2 processors all the time, will be slowed to a lesser extent. In fact, LO-MG uses the full array of processors only for fine grid computations, i.e. $\frac{1}{log(m)}$ of the time. Based on a remark of M. Hyman, we note the following interesting hybrid method. Assume we have $\frac{4}{3}p^2$ processors to solve an m by m, $m>p$ problem. Use the LO-MG method on grid levels $log(p)+1$ through $log(m)$ and use C.I. for levels at or below $log(p)$. It can be shown that the computation can be arranged so that all communication on grid levels above $log(p)$ is between nearest neighbors and on the lower levels any of the communication models of section 4 may be applied.

An alternative approach in the limited processor case is to consider locally refined grids. In a companion paper, [11] we show that there is a locally refined version of the concurrent iteration algorithm that can use the high levels of concurrency to overlap pipelined communication with computation. This paper also compares the performance of a systolic band solver to that of the multigrid algorithms. It is shown that on small problems the direct solver is competitive, but for large system the iterative methods seem to be superior.

The second area not directly considered here is the comparison of direct and iterative methods. In the case of 2 dimensional problems of the size of those considered in section 5, a band systolic solver [18] will work in approximately $8m^2$ steps. To derive a similar estimate for concurrent iteration consider table 5.1 and 5.2. The number of grid levels in a problem of size m is $log(m)$ and the constant cost model grows roughly linearly in the number of levels. An upper bound of $40log(m)$ for the figures in the table multiplied by the time to complete one fine grid relaxation R yields an estimate of approximately $40R*log(m)$. If we assume the communication model is the log cost model with one unit of communication equal to one arithemitic operation, then asymptotically the time is quadradic in the number of grid levels. A generous upper bound is given by $20log^2(m) + 40R*log(m)$. If R, the execution time for one relaxation, is around 20 steps for arithimetic and communication overhead and the grid size $m=64$, then C.I. in the log model runs in time 5520 vs. 16384 for the the direct method. This implies that if the approximations above hold for larger values of m, the iterative scheme is substantially faster in spite of the extra communication costs.

For 3 dimensional problems, the situation is even more dramatic. The band solver requires roughly m^4 processors for the band width m^2 system based on a m by m by m cube. The computation time jumps to $O(m^3)$. On the other hand, the multigrid algorithms are relatively blind to the dimension of the system. The number of processors jumps to $O(m^3)$ but the time complexity should not change much from the estimates above: the value of R will go to around 60 and the rates of convergence (reflected in the constants 20, 40) will increase at most only slightly. The organization of the parallel algorithms and data structures for realistic 3 dimensional problems remains an important open area of research.

5. Acknowledgments

The authors would like to acknowledge the encouragement and helpful criticism provided by Robert Voigt of ICASE and the support of ICASE and the National Science Foundation.

6. References

[1] L. Adams and J. M. Ortega, "Multi-color SOR Method for Parallel Computation," ICASE Report No. 82-9, April 8, 1982.

[2] R. E. Bank, T. Dupont, "An optimal order process for solving elliptic finite element equations," MATH COMP 36, 35-51 (1980).

[3] R. E. Bank, A. H. Sherman, "Algorithmic Aspects of the Multi-Level Solution of Finite Element Equations", CNA-144, Center for Numerical Analysis, The University of Texas at Austin, Austin, Texas, 1978.

[4] K. Batcher, "MPP - A Massively Parallel Processor," Proceedings of the 1979 International Conference on Parallel Processing, pp.249.

[5] G. Baudet, "Asynchroneous Iterative Methods for Multiprocessors," *J. Assoc. Comp. Mach.* 25, pp226-242.

[6] D. Braess, "The convergence rate of a multigrid method with Gauss-Seidel relaxation for the Poisson Equation," *Proc. Multigrid Meth. Conf.* , Nov. 1981, Cologne, W. Germany, Springer-Verlag Lecture Notes in Math. (W. Hackbush and U. Trottenberg eds.), Berlin 1982.

[7] A. Brandt, "Multigrid Solvers on Parallel Computers", ICASE NASA Langley Research Center, Hampton, VA, ICASE Report No. 80-23, 1980.

[8] A. Brandt, "Multilevel Adaptive Solutions to Boundary Value Problems," *Math Comp.* 31, 1977, 333-391.

[9] R. P. Federenko, "The Speed of Convergence of an Iteration Process," ZH. vychisl. Mat. mat. Fiz. 4, (Russian), pp. 559-564, 1964.

[10] D. B. Gannon, "A Note on Pipelining a Mesh Connected Multiprocessor for Finite Element Problems by Nested Dissection", Proceedings of the 1980 International Conference on Parallel Processing, IEEE Cat. no. 80CH1569-3, pp.197-204.

[11] D. B. Gannon, "On Mapping Non-uniform P.D.E. Structures and Algorithms onto Uniform Array Architectures," Proceedings of the 1981 International Conference on Parallel Processing, pp.100-106.

[12] D. B. Gannon and J. Van Rosendale, "Solving Elliptic P.D.E. Problems on Parallel Processors: Experiments with Locally Refined Grids," Technical

Report, ICASE Nasa Langley Research Center, Hampton Va., 1982.

[13] C. Grosch, "The Effect of the Data Transfer Pattern of an Array Computer on the Efficiency of Some Algorithms for the Tri-Diagonal and Poisson Problem", Array Architectures for Computing In the 80's and 90's, ICASE Workshop, April 1980, Hampton, Virginia.

[14] W. Hackbusch, "On the Multigrid Method Applied to Difference Equations," Computing 20, pp. 291-306, 1978.

[15] W. Hackbusch, "Multigrid Convergence Theory," Proceedings of the 1981 Conference on Multigrid Methods, Springer-Verlag, Lecture Notes in Mathematics, 1982.

[16] H. Jordan, "A Special Purpose Architecture for Finite Element Analysis", Proceedings of the 1978 International Conference on Parallel Processing, pp 263-266.

[17] R. N. Kapur, U. V. Premkumar, G. J. Lipovski, "Organization of the TRAC Processor-Memory Subsystem," AFIPS Conf. Proc. pp. 632-629, May 1980.

[18] H. T. Kung, C. E. Leiserson, "Algorithms for VLSI Processor Arrays", C. Mead and L. Conway, Introduction to VLSI Systems, Addison-Wesley, Reading, Ma., (1980). pp. 271-292.

[19] D. H. Lawrie, "Access and Alignment of Data in an Array Processor," IEEE Trans. on Computers, Vol. C-24, No. 12, pp. 1145-1155, Dec. 1975.

[20] T. Leighton, D. Klietman, M. Lepley, G. Miller, "New layouts for the Shuffle-Exchange Graph," STOC(Milwaukee 1981), 278-292.

[21] J. W. H. Liu, "The solution of Mesh Equations on a Parallel Computer," Dept. of Computer Science, Univ. of Waterloo, Waterloo, Ontario, Report CS-78-19, 1978.

[22] S. F. Lundstrom, G. H. Barnes, "A Controllable MIMD Architecture", Proceedings of the 1980 International Conference on Parallel Processing, IEEE Cat. no. 80CH1569-3, pp.19-27.

[23] S. McCormick, "MultiGrid Methods for Variational Problems: the V-cycle," Proc. IMACS World Cong. Sys. Sym. Sci. Comp., Aug. 8, 1982, Montreal, Canada.

[24] R. A. Nicolaides, "On the L2 Convergence of an Algorithm for Solving Finite Element Equations", Math. Comp. 31, 1977, 892-906.

[25] A. H. Sameh, S. C. Chen, and D. J. Kuck, "Parallel Poisson and Biharmonic Solvers", Computing 17 (1976), 219-230.

[26] J. Schwartz, "Ultracomputer", ACM Transactions on Programming Languages and Systems, 1981.

[27] H. J. Siegel, et. al., "An SIMD/MIMD Multimicroprocessor System for Image Processing and Pattern Recognition", IEEE Conf. on Pattern. Recog. Image Proc. Aug. 1979, pp. 214-224.

[28] Snyder, L., A Synopsis of the Blue CHiP Project, Dept. of Computer Sciences, Purdue University, Report 1980.

[29] O. Storaasli, F. Peebles, T. Crockette, J.Knott, L. Addams, "The Finite Element Machine: An Experiment in Parallel Processing," NASA Technical Memorandum no.84514 July, 1982.

[30] J. R. Van Rosendale, "Rapid Solution of Finite Element Equations on Locally Refined Grids by Multi-Level Methods", Department of Computer Science, University of Illinois, UIUCDCS-R-80-1021, Urbana, Illinois, 1980.